Homework 1

Due by **11:59pm, Thursday, September 13, 2018**.

Make sure you follow all the homework policies (../../policies/hw-policy.html).

All submissions should be done via Autolab (../../autolab.html).

1. Just in there is any confusion: this is a totally made up scenario: I'm not claiming the ability to predict the future.

Some Questions on Stable Matching

Sample Problem

The Problem

Decide whether the following statement is true or false:

In every Stable Marriage problem instance where a man m and woman w have each other as their least preferred partner, the following is true. There is no stable matching for the instance where (m, w) are matched.

If you state true then you will have to formally argue why the statement is correct. If you state false, then you have to give a counter-example.

Click here for the Solution

More Practice Questions

The Problem

Exercises 1 and 2 in Chapter 1.

Submission

You will **NOT** submit this question. This is for you to get into thinking more about the stable matching problem.

Question 1 (Programming Assignment) [30 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

Given an input for the stable matching problem, output **all** stable matchings for that instance.

Input

The input is an instance of the stable marriage problem in a text file of the following format:

```
n                                 <- Number of men and women (each)
m_1^1  m_2^1  m_3^1 ...  m_n^1    <- Preference of the 1st woman (most prefer
m_1^2  m_2^2  m_3^2 ...  m_n^2    <- Preference of the 2nd woman (most prefer
m_1^3  m_2^3  m_3^3 ...  m_n^3    <- Preference of the 3rd woman (most prefer
m_1^4  m_2^4  m_3^4 ...  m_n^4    <- Preference of the 4th woman (most prefer
.
.
.
m_1^n  m_2^n  m_3^n ...  m_n^n    <- Preference of the nth woman (most prefer
w_1^1  w_2^1  w_3^1 ...  w_n^1    <- Preference of the 1st man (most preferr
w_1^2  w_2^2  w_3^2 ...  w_n^2    <- Preference of the 2nd man (most preferr
w_1^3  w_2^3  w_3^3 ...  w_n^3    <- Preference of the 3rd man (most preferr
w_1^4  w_2^4  w_3^4 ...  w_n^4    <- Preference of the 4th man (most preferr
.
.
.
w_1^n  w_2^n  w_3^n ...  w_n^n    <- Preference of the nth man (most preferr
```

For example

```
4                       <- Number of men and women (each)
3   4   2   1           <- Preference of the 1st woman (most preferre
2   4   3   1           <- Preference of the 2nd woman (most preferre
2   3   4   1           <- Preference of the 3rd woman (most preferre
4   1   2   3           <- Preference of the 4th woman (most preferre
1   2   3   4           <- Preference of the 1st man (most preferred
3   1   4   2           <- Preference of the 2nd man (most preferred
2   1   4   3           <- Preference of the 3rd man (most preferred
4   1   2   3           <- Preference of the 4th man (most preferred
```

Output

The output is the set of **all** stable matchings for the input in a text file of the following format:

```
x                                              <- Number of stable matchings 
[(m1^1,1),(m2^1,2),(m3^1,3),(m4^1,4),...,(mn^1,n)]      <- Pairing of the form (m,w
[(m1^2,1),(m2^2,2),(m3^2,3),(m4^2,4),...,(mn^2,n)]      <- Each stable matching is 
[(m1^3,1),(m2^3,2),(m3^3,3),(m4^3,4),...,(mn^3,n)]
.
.
.
[(m1^x,1),(m2^x,2),(m3^x,3),(m4^x,4),...,(mn^x,n)]      <- xth stable matching
```

Please note that each stable matching outputted is sorted by women first. For example:

```
3                               <- Number of stable matchings found
[(3,1),(1,2),(2,3),(4,4)]       <- Pairing of the form (m,w)
[(4,1),(2,2),(1,3),(3,4)]
[(4,1),(3,2),(1,3),(2,4)]
```

Please note that the above output is an example is not the correct solution to the input example given above. For correct pairs of input and output, please look at any of the code skeletons.

Hint

The best possible algorithm for this problems that we are aware of runs in time $O(n^2 \cdot n!)$. This is not terribly efficient but an exponential runtime for this problem is unavoidable-- see Question 3. For this reason we will be testing your implementations with testcases on $n \leq 10$.

It *might* be useful to implement Heap's algorithm (../../../support/proofs/index.html) as one of the functions that your algorithm implementation uses.

! Note

Both the input and output parsers in each of the three languages are already written for you. Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file). Also note that you do not have to sort your output: we'll take care of that.

! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Java Python C++

Download Java Skeleton Code (HW1Java.zip)

Directory Structure

```
├── src
│   ├── ub
│       ├── cse
│           ├── algo
│               ├── Driver.java
│               ├── HW1Utility.java
│               ├── Marriage.java
│               ├── Matching.java
│               ├── PreferenceLists.java
│               ├── Solution.java
├── testcases/
│   ├── input1.txt
│   ├── input2.txt
│   ├── input5.txt
├── outputs/
│   ├── output1.txt
│   ├── output2.txt
│   └── output5.txt
```

You are given six coding files: `Driver.java` , `HW1Utility.java` , `Marriage.java` , `Matching.java` , `PreferenceLists.java` and `Solution.java` . `Driver.java` takes the input file, parses it with a new instance of `HW1Utility` and creates an instance of the class `Solution` and calls the `outputStableMatchings()` method on it. It then prints all the computed perfect matchings (which, if your code is correct, will all be stable matchings). You only need to update the `Solution.java` file.

Hint

You're also given a helper method `permutate` in `Solution.java` that generates and prints out all the permutations (see below for one way to call this helper function). This could be useful in your solution (after some modification).

On top of that, you may write your own helper methods and data structures in it.

The testcases folder has 3 input files and their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding.

Method you need to write:

```
/**
* This method must be filled in by you. You may add other meth
* but they must remain within the Solution class.
* @return Your set of stable matches. Order does not matter.
*/
public ArrayList outputStableMatchings() {
    /* The code below just calls the permutate function, whicl
    ArrayList<Marriage> mar = new ArrayList();
    for(int k = 1; k<=numberOfMenAndWomen; ++k) {
        Marriage m = new Marriage(k, k);
        mar.add(m);
    }
    Matching mat = new Matching(mar);
    System.out.println("Printing all possible perfect matchin
    System.out.println("---------------------------");
    permutate(mat,numberOfMenAndWomen);
    System.out.println("---------------------------");
    /*permuate call done*/

    return stableMatchings;
}
```

The `Solution` class has 5 instance variables.

- `numberOfMenAndWomen` which is of type int and stores n.
- `men` which is of type `HashMap<Integer, ArrayList<Integer>>` and stores the preference lists of men. Please note that the front of the `ArrayList<Integer>` (index 0) denotes the most preferred partner.
- `women` which is of type `HashMap<Integer, ArrayList<Integer>>` and stores the preference lists of women. Please note that the front of the `ArrayList<Integer>` (index 0) denotes the most preferred partner.
- `stableMatchings` which is of type `ArrayList<Matching>` and stores the set of stable matchings you find for the given instance.
- `count` which is of type int and used for the helper method `permutate` .

The Other files

`Marriage.java` defines the `Marriage` class. This should be fairly intuitive. Below is the entire content of the class:

```java
public class Marriage implements Comparable{
    public Integer man;
    public Integer woman;
    Marriage(Integer man, Integer woman){
        this.man = man;
        this.woman = woman;
    }
    @Override
    public boolean equals(Object obj){
        return (man.equals(compare.man)) && (woman.equals
    }
    @Override
    public String toString(){
        return "(" + man + ", " + woman + ")";
    }
    @Override
    public int compareTo(Marriage other){
        return this.woman.compareTo(other.woman)
    }
}
```

The file `HW1Utility.java` handles some of the background stuff: e.g. `readFile` reads in the file passed as a command line argument to `Driver.java` and populates the preference lists within the `HW1Utility` class.

Compiling and executing from command line:

Assuming you're in the same directory level as `src` . Run `javac src/ub/cse/algo/*.java` to compile.

To execute your code on `input1.txt` , run `java -cp "src" ub.cse.algo.Driver testcases/input1.txt` .

Submission

You only need to submit `Solution.java` to Autolab.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

Question 2 (Stable TV Show Schedules) [45 points]

The Problem

It is the year 2035 and only two online video content platforms have survived the frenzied competition to get people to watch their ``TV" online. [1] In the future, `Amazing` and `Bombastic` are the two surviving platforms. As is the case now, both these platforms release all the episodes of a new show on the same day and in the future the consumer has been trained to watch all of the episodes in one sitting. However, `Amazing` (henceforth just called A) and `Bombastic` (henceforth just called B) are still competing with each other and they have to figure out how to schedule the release dates of their new shows for fall of 2035.

In particular, A and B have n shows in their roster that they want to schedule on the same n release dates.

Since this is 2035 both A and B have the entire TV show preferences of every human being on the planet, given a show m it can accurately predict a *rating* for m. You can assume that the rating is an integer and a larger rating the larger number of viewers will watch that show (and hence make more money). We say for a given release date a platform *wins* the release date if its released show on that release date has a higher rating than any other show released on the same day.

What about ties?

If on a given date both platforms release shows with the *same* rating, then you should assume that A wins.

At the beginning of each fall, A reveals a schedule S and B reveals a schedule T. A schedule for a platform assigns each of its n shows to the n release dates. Thus, given the schedules S and T, each release date has exactly two shows being released on that date: one from A and another from B. The goal of the studios in designing the schedules S and T is to win on as many release dates as possible.

In this problem, you will explore the notion of stability in this context. In particular, a given pair of schedule (S, T) is said to be *stable* if no platform can unilaterally change its own schedule and end up winning on more release dates. More precisely, the pair of schedule (S, T) is stable if for any alternate set of schedules S' for A, A wins on no more release dates with the pair of schedules (S', T) than with the pair (S, T). Further, for any other schedule T' for B, B wins on no more release dates with the pair of schedules (S, T') than with the pair (S, T).

Here is an example to illustrate the above. Consider the case of $n = 3$ and A has show/rating pairs of $(m_1, 100), (m_2, 101), (m_3, 102)$ while B has pairs $(m_4, 1), (m_5, 2), (m_6, 3)$, where each m_i is a show and the corresponding number is its rating. In this case *every* pair of schedule (S, T) is stable (since A wins on all the release dates irrespective of the schedule).

The corresponding question to one we asked in class is the following: For **every** set of n movies and their corresponding success numbers, does there always **exists** a stable pair of schedules?

Here are the two parts:

- Part `(a)` : Show that the answer is **yes for the following special case**: the lowest rating of any show in A is strictly larger than the largest rating of any show in B.
- Part `(b)` : Show that the answer to the question above is **no for the general case**. In other words show that there exists an example of shows and their corresponding ratings, for which there does not exist any stable pair of schedules.

Note

It is helpful for part `(b)` , to formally write down in first order logic, what the problem is asking for. To help you guys out, here is an overview of what you need to do: You have to present only **one** input and then prove that **every** possible pair of schedules (S, T) is ***NOT* stable**.

Hint

Note that for any given n, there are $(n!)^2$ many possible pairs of schedules (S, T).

Common Mistake

A common mistake for part `(b)` is to present an example and then argue that some *but not all* $(n!)^2$ schedules are not stable.

Walkthrough video

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

! PDF only please

Autolab might not be able to display files in formats other than PDF (e.g. Word cannot be displayed). **If Autolab cannot display your file, then you will get a zero (0) on the entire question. Note that Autolab will NOT give an error message if you submit non-PDF file, so it is YOUR responsibility to make sure you submit in the correct format.** Also the file size has to be **at most 3MB**.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part `(a)` of this problem:

1. `Proof idea` : `10` points.

and here is the high level grading rubric for part `(b)` :

1. `Proof idea` : `17` points for a counterexample idea explaining the insight behind why you think the property does not holds.
2. `Proof details` : `18` points for a complete description of a counterexample *and* a complete proof for why the given counter example does not have any stable schedule.

! Note

If you do not have separated out proof idea and proof details for part `(b)` , you will get a zero(0) irrespective of the technical correctness of your solution..

Templates

Download LaTeX template. (latex-template.zip)

Download Microsoft Word template. (word-template.zip)

! Note

Your must explicitly list your sources and collaborators when you upload your submission to Autolab. Note that there are only five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. If you did not consult any source or did not collaborate with anyone just say `None` .

Question 3 (Many Stable Matchings) [25 points]

The Problem

In class we will shortly see that every stable matching instance has at least one stable matching. We have also seen a stable matching instance that has two stable matchings. In this problem, you will be asked to come up with stable matching instance that have (many) more stable matchings. Here are the two parts:

- Part `(a)` : Show that for every large enough n, there is a stable matching instance on n men and n women such that the instance has at least $\Omega(n)$ distinct stable matchings.
- Part `(b)` : For every $n \geq 3$ that is a multiple of 3, show that there is a stable matching instance on n men and n women such that the instance has at least $3^{n/3}$ distinct stable matchings.

To get **full credit**, you should be able to present an instance for *every* $n \geq 3$ that is a multiple of 3.

Hint

First try and construct such an instance for $n = 3$. Try and extend the instance to every n that is a multiple of 3.

Note

If you show a family of instances with (many) more stable matchings than $3^{n/3}$ that is of course fine. In fact, if you are looking for a puzzle, try and prove as large a lower bound on the number of stable matching as you can. If you get something better than say 3^n come talk to me. **Warning**: Showing such a bound will solve an open problem, so be warned that doing so might not be easy.

Walkthrough video

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

! PDF only please

Autolab might not be able to display files in formats other than PDF (e.g. Word cannot be displayed). **If Autolab cannot display your file, then you will get a zero (0) on the entire question. Note that Autolab will NOT give an error message if you submit non-PDF file, so it is YOUR responsibility to make sure you submit in the correct format.** Also the file size has to be **at most 3MB**.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part `(a)` of this problem:

1. `Proof idea`: `10` points.

and here is the high level grading rubric for part `(b)`:

1. `Proof idea`: `7` points for outlining the proof idea.
2. `Proof details`: `8` points for the proof of the general case (i.e. for every $n \geq 3$ that is a multiple of 3).
3. `Note`: If your solution only presents example(s) for some *specific* $n \geq 3$, then you will gt **no points**.

! Note

If you do not have separated out proof idea and proof details for part `(b)`, you will get a zero(0) irrespective of the technical correctness of your solution..

Templates

Download LaTeX template. (latex-template.zip)

Download Microsoft Word template. (word-template.zip)

! Note

Your must explicitly list your sources and collaborators when you upload your submission to Autolab. Note that there are only five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. If you did not consult any source or did not collaborate with anyone just say `None`.